

# Introducing Ruby

Charles Severance

Textbook: Build Your own Ruby on Rails Application by Patrick Lenz (ISBN:978-0-975-8419-5-2)

This is a “firehose” lecture...

# Users .vs. Programmers

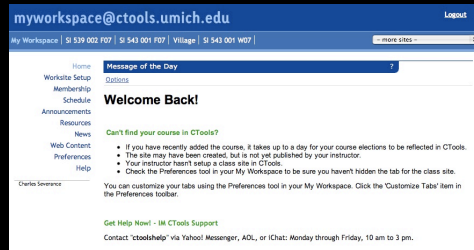
- Users see computers as a set of tools - word processor, spreadsheet
- Programmers have some tools that allow them to build new tools
- Programmers sometimes write tools for lots of users and sometimes programmers write little widgets for themselves to automate a task

# What is Code?

- A set of stored instructions
  - It is a little piece of our intelligence in the computer
  - It is a little piece of our intelligence we can hand out

# Outline

- Teach 8 weeks of Computer Programming in 1.5 hours
- Look at some simple samples at the end of the lecture
- Review as needed for the reset of the semester



User



Creators



From a software creator's point of view, we build the software. The end users (stakeholders/actors) are our masters - who we want to please - often they pay us money when they are pleased. But the data, information, and networks are our problem to solve on their behalf. The CPU and Memory are our friends and allies in this quest.

# Object Oriented Approach

# Object Oriented

- Nothing “magical” - A way or organizing and decomposing large / complex applications
- Principles
  - Information hiding
  - Divide and Conquer
  - Code Reuse - Share useful components



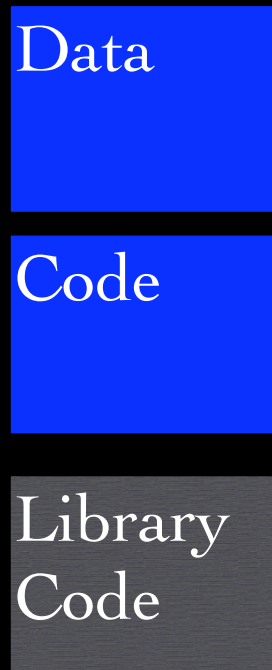
# Ruby and Object Orientation

- Ruby is 100% Object Oriented - “burn the ships”
- Ruby is designed from the ground up to “assume OO”
- Since OO is the only way to do anything in Ruby - it is also quite natural

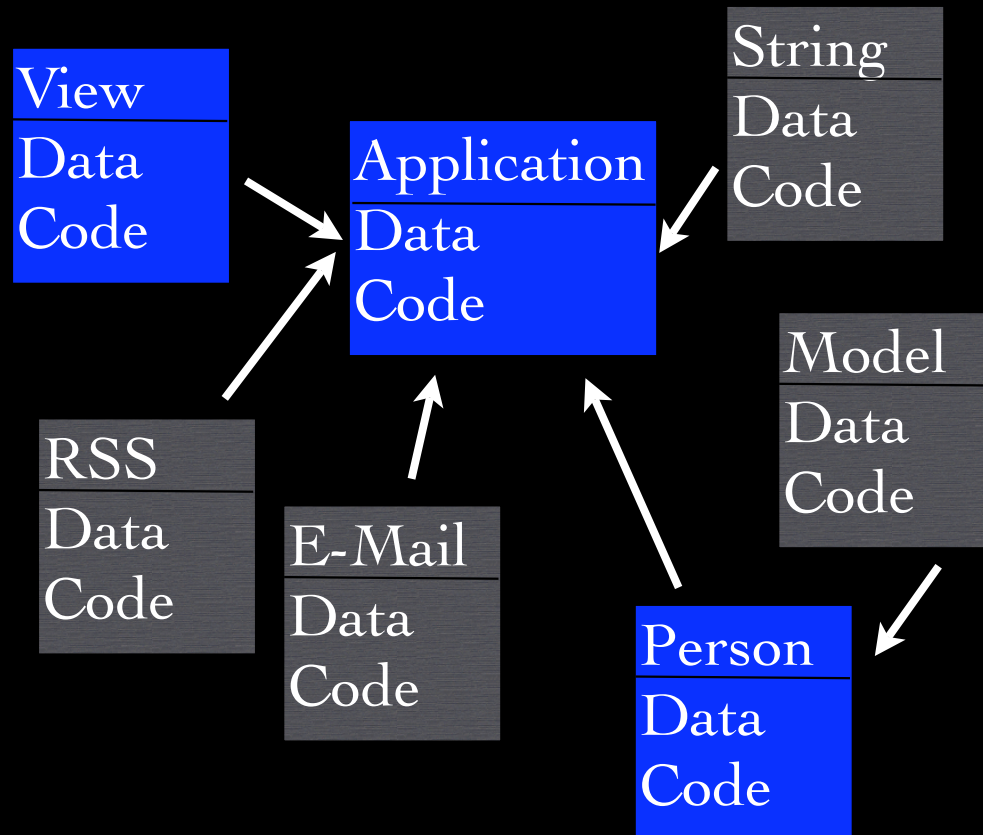
# Object Oriented - Not!

- Before we define what \*is\* object-oriented programming - lets define how we programmed in the pre-object days in Languages like C and Fortran
- In the good old days - there were Programs (Code) and Data
- Sometimes we had Code Libraries which were reusable
- Sometimes we grouped pieces of data to make a complex data type - Person = First\_name + Last\_name

## Pre-Object Oriented



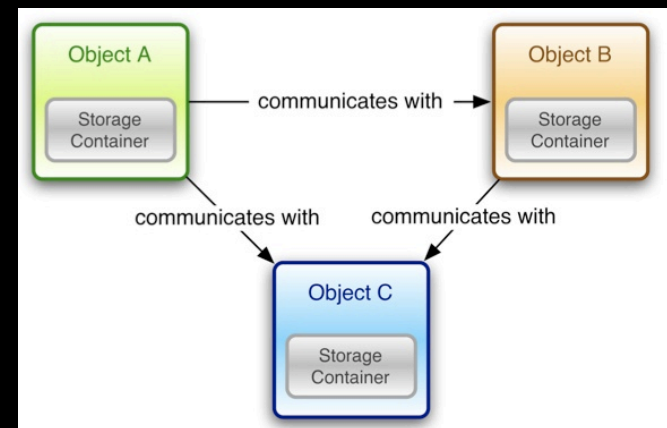
## Object Oriented



Information Hiding, Divide and Conquer, Code Reuse

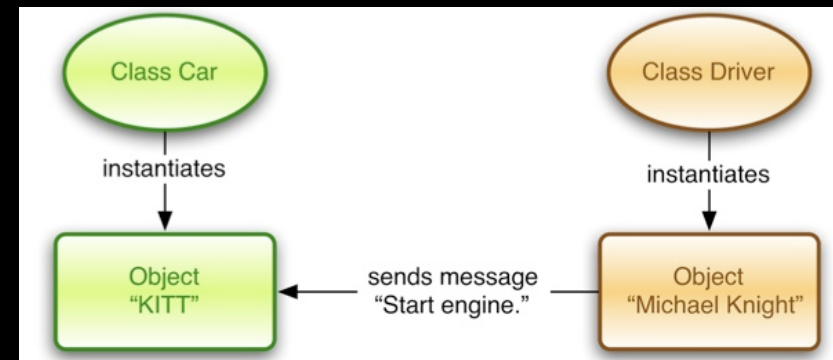
# Objects Working Together

- Our job is to marshall and control objects and to accomplish what the user wants and needs, and/or is willing to pay for

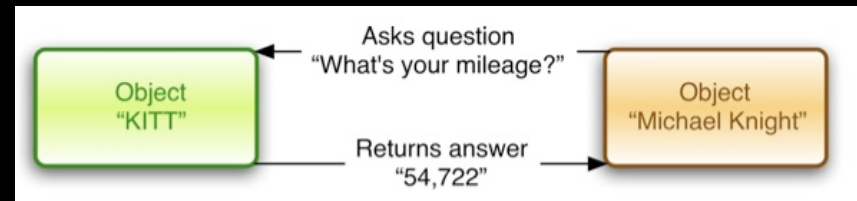


# Classes and Objects

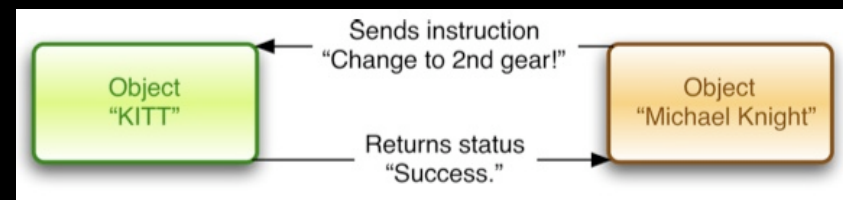
- Classes are the templates
  - There is one “class”
  - We define the class
- Objects are the instances
  - There can be many instances
  - We create and use objects to get work done
- Sometimes we make a class and something else creates and uses the object - often we must follow rules



# Messages (or Methods)



- We send “messages” to Objects to cause some effect
- These messages usually trigger some code to be executed within the object
- Messages can have parameters

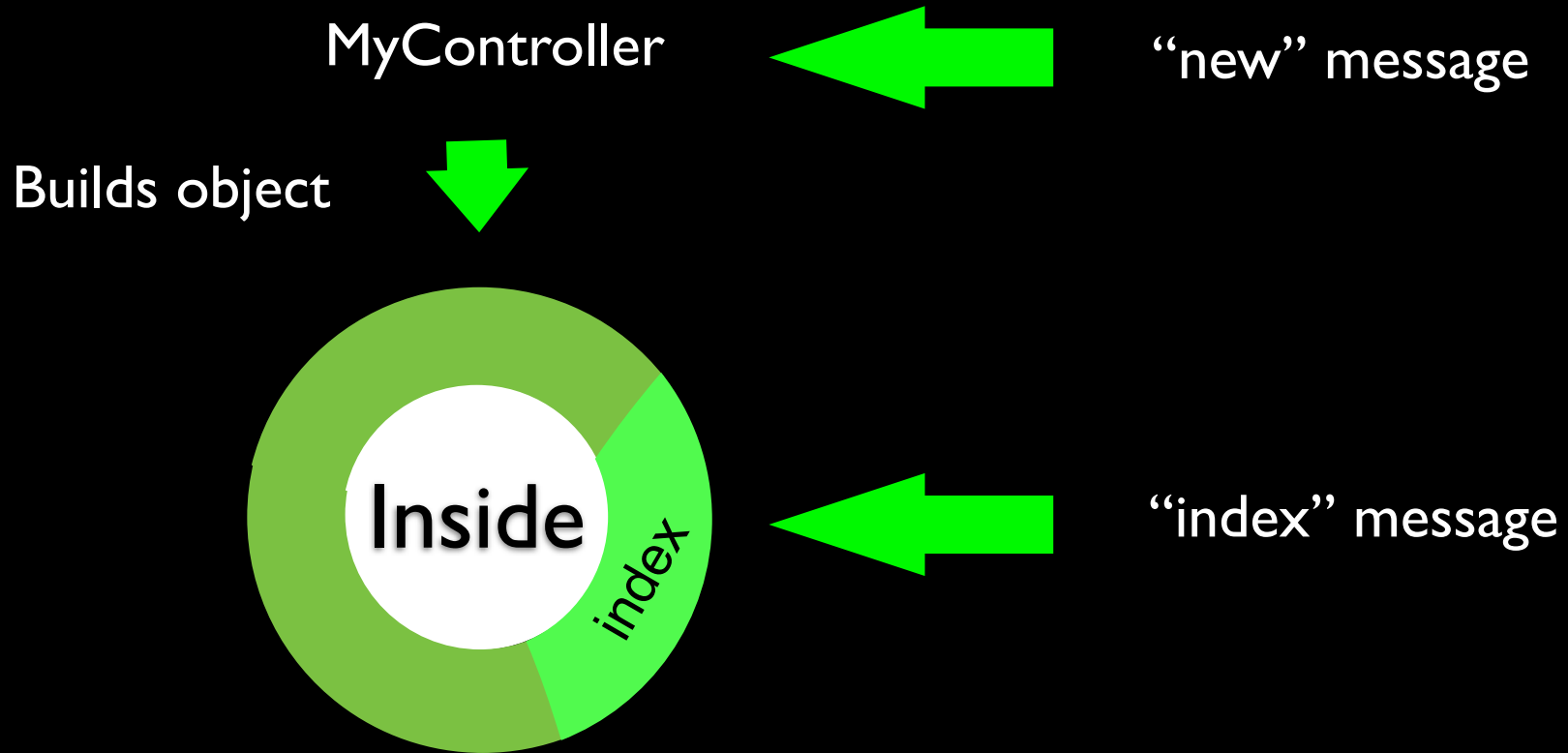


# The View From Inside

```
class MyController < ApplicationController
  # My Data
  def index
    # My Code
  end
end
```

# The View From Outside

```
# Time to fire up Controller
c = MyController.new
c.index # Here is your Message
```



Information Hiding



# Clear Contracts - Interfaces

- When the contract between Objects is very very complete and clear - work can proceed very independently - neither side need to be aware of the complexity or issues on the other side.
- This contract is called an “interface”
- “If you are going to be a fully compliant Rails controller - you must implement this interface.”

```
class GuessController < ApplicationController

  def index
  end

  def try
    logger.info(params[:guess])
    if params[:guess] == "food"
      @result = "correct"
    else
      @result = "incorrect"
    end
  end
end

end
```

# The Ruby Language

# Ruby vs. Rails

- Ruby
  - Is a programming language - similar to all other programming languages
  - Variables, loops, functions, iteration, etc
- Rails
  - Is a framework written in Ruby used to develop web applications
  - Databases, Request Response Cycle, Parameters, HTML templates

# Ruby is an Interpreted Language

- Compiled Languages: FORTRAN, C, C++, Pascal, COBOL, ADA
- Interpreted Languages: SmallTalk, PHP, Basic, Perl, Ruby, Python
- Mostly Compiled: Java

# Advantages Both Ways

- Classic Arguments
  - Compiled Languages: Faster, Less Memory,
  - Interpreted languages: More Flexible, More productive
- Interpreted languages increasingly use “Just in time” or “Run-time” compilation or partial compilation techniques to get speed
- As microprocessors get faster and faster, fewer and fewer applications are written in compiled languages

# Interactive Ruby Interpreter

- Great way to explore the Ruby language (not Rails)
- `irb --simple-prompt`
- <http://tryruby.hobix.com/>
- Google: interactive ruby

Prompt

```
$ irb --simple-prompt
>> i = 1
=> 1
>> i = i + 5
=> 6
>> i = i + 5
=> 11
```

Output from IRB

You type Ruby

try ruby! (in your browser)

http://tryruby.hobix.com/ rails ApplicationConti

CT Sakai Collab Source Zam Confluence Bugs sc1 scoot Cafe Pluto uP2 S:8080 PDA Desiderata

# Try Ruby!

a hands-on tutorial

**Interactive ruby ready.**

```
>> i = 1
=> 1
>> i = i + 5
=> 6
>> i = i + 5
=> 11
>>
```

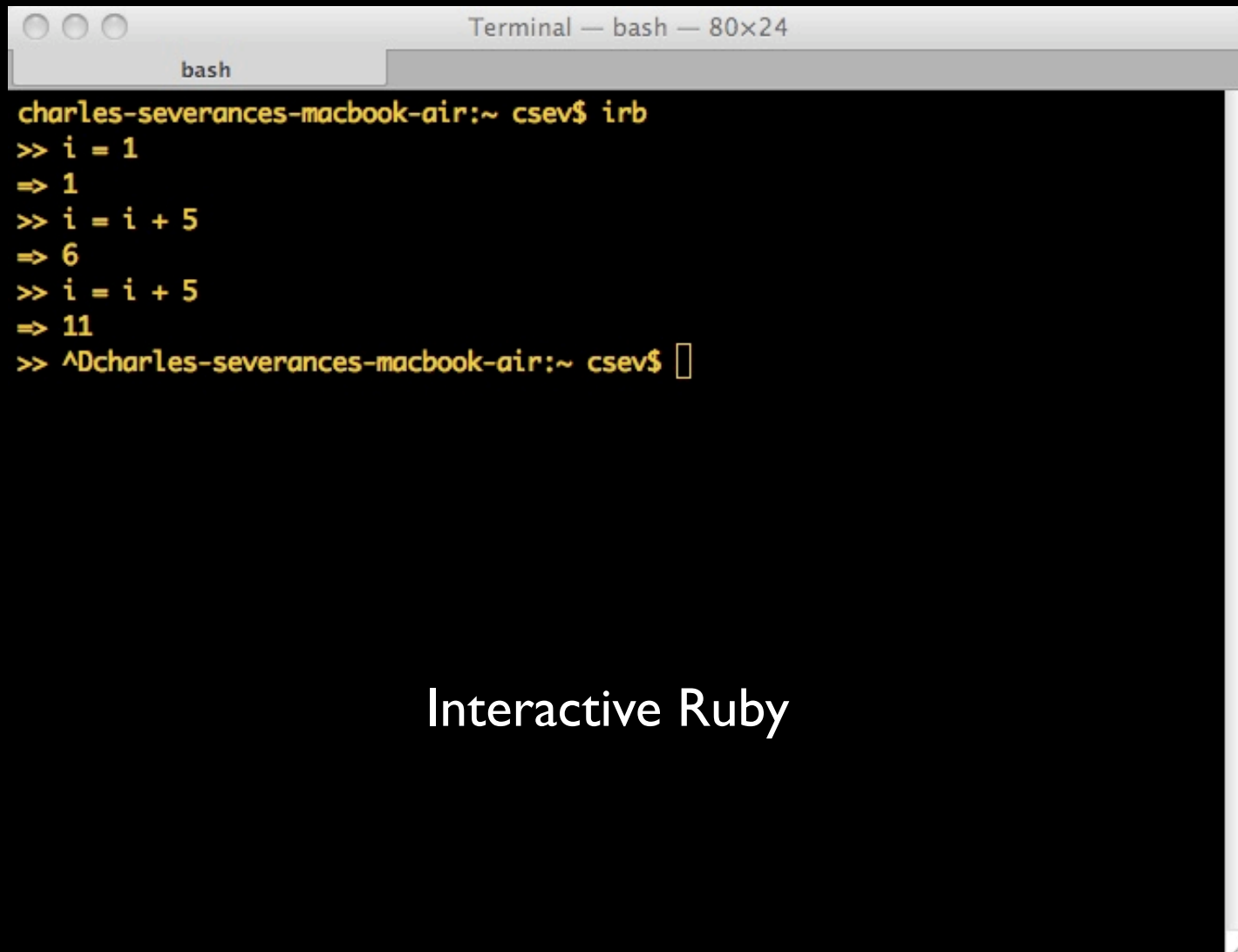
## Got 15 minutes? Give Ruby a shot right now!

Ruby is a programming language from Japan (available at [ruby-lang.org](http://ruby-lang.org)) which is revolutionizing the web. The beauty of Ruby is found in its balance between simplicity and power.

Try out Ruby code in the prompt above. In addition to Ruby's builtin methods, the following commands are available:

<b>help</b>	Start the 15 minute interactive tutorial. Trust me, it's very basic!
<b>help 2</b>	Hop to chapter two.
<b>clear</b>	Clear screen. Useful if your browser starts slowing down. Your command history will be





```
Terminal — bash — 80x24
bash
charles-severances-macbook-air:~ csev$ irb
>> i = 1
=> 1
>> i = i + 5
=> 6
>> i = i + 5
=> 11
>> ^Dcharles-severances-macbook-air:~ csev$
```

## Interactive Ruby

```
C:\ Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\chuck>cd \ruby

C:\ruby>rsu
Ruby added to path - C:\ruby\bin
Switching to ruby_apps directory
=====
Volume in drive C has no label.
Volume Serial Number is 985E-083F

Directory of C:\ruby\rails_apps

09/11/2007  10:52 AM    <DIR>          .
09/11/2007  10:52 AM    <DIR>          ..
09/05/2007  03:08 AM    <DIR>          app0
09/11/2007  10:52 AM    <DIR>          assn2
               0 File(s)                0 bytes
               4 Dir(s)  4,393,693,184 bytes free

C:\ruby\rails_apps>irb
irb(main):001:0> i = 1
=> 1
irb(main):002:0> i = i + 5
=> 6
irb(main):003:0> i = i + 5
=> 11
irb(main):004:0>
C:\ruby\rails_apps>
```

# Ruby Punctuation

- # indicates a comment
- . (period) indicates a message / method call
- Parameters to method calls are enclosed by parenthesis and separated by commas - if there are no parameters parenthesis are optional
- “ or ‘ starts and stops strings

# The View From Inside

```
class MyController < ApplicationController
  # My Data
  def index
    # My Code
  end
end
```

# The View From Outside

```
# Time to fire up Controller
c = MyController.new
c.index # Here is your Message
```

# White Space and Line Ends

- White space does not matter - but indenting is greatly appreciated - if you don't indent your code - you are showing yourself to be an uncultured swine
- Line ends do matter where they make sense
- Technically, Ruby statements end with a semi-colon - which are optional as long as you are reasonable

## Statement Skillz

Two statements  
on one line  
semicolon  
required

One Fish, Two Fish,  
Red Fish, Blue Fish...

```
irb --simple-prompt
```

```
>> i = 1
```

```
=> 1
```

```
>> i = 2
```

```
=> 2
```

```
>> x = 3 x = 4
```

```
(irb):3: syntax error
```

```
x = 3 x = 4
```

```
^
```

```
from (irb):3
```

```
>> x = 3 ; x = 4
```

```
=> 4
```

```
>> i = 2 +
```

```
?> 3
```

```
=> 5
```

```
>>
```

Two statements  
on two lines

Weak try at two  
statements on  
one line.

Denied by  
the IRB!

One statement  
on two lines

# Literal Objects

- FixNum 1 2 3
- String “Current Speed”
- Float 6.02
- Symbol :thing
- nil - means “no object here”

Constants are objects and have methods....

```
irb --simple-prompt
>> 2.class
=> Fixnum
>> 3.14.class
=> Float
>> "Hello".class
=> String
>> 'X'.class
=> String
>> :fred.class
=> Symbol
>> nil.class
=> NilClass
```

# CONSTANTS

- If a variable name is upper case it is viewed as a constant that once set should not be changed

```
$ irb --simple-prompt
```

```
>> X = 5
```

```
=> 5
```

```
>> X = 2
```

```
(irb):2: warning: already  
initialized constant X
```

```
=> 2
```

```
>>
```



# What does “Type” Mean?

- In Ruby variables, literals, and constants have a “type”
- Ruby knows the difference between an integer number and a string
- For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>> dd = 1 + 4  
=> 5  
>> ee = "hello " + "there"  
=> "hello there"  
>>
```

# Type Matters

- Ruby knows what “type” everything is
- Some operations are prohibited
- You cannot “add 1” to a string
- We can ask Ruby what type something is by using the `class` method

```
>> eee = "hello " + "there"
```

```
=> "hello there"
```

```
>> ee = ee + 1
```

```
TypeError: can't convert Fixnum  
into String
```

```
from (irb):4:in `+'
```

```
from (irb):4
```

```
>> ee.class
```

```
=> String
```

```
>> "hello".class
```

```
=> String
```

```
>> 1.class
```

```
=> Fixnum
```

```
>>
```

```
irb> # This is a comment. It doesn't actually do anything.  
irb> | # So is this, but this one comes after a statement.  
=> |  
irb> fox = "The quick brown fox"    # Assign to a variable  
=> "The quick brown fox"  
irb> fox.class                      # Display a variable's class  
=> String  
irb> fox.length                     # Display a variable's length  
=> 19
```

```
irb> # Optional Parenthesis
```

```
irb> fox.class()
```

```
=> String
```

```
irb> fox.class
```

```
=> String
```

```
irb> # Messages with more than one parameter
```

```
irb> "jumps over the lazy dog".insert(0, 'The quick brown fox ')
```

```
=> "The quick brown fox jumps over the lazy dog"
```

Google: ruby string class

<http://www.ruby-doc.org/core/classes/String.html>

```
irb> fox.upcase  
=> "THE QUICK BROWN FOX"  
irb> fox  
=> "The quick brown fox"  
irb> fox.upcase!  
=> "THE QUICK BROWN FOX"  
irb> fox  
=> "THE QUICK BROWN FOX"  
irb> fox.empty?  
=> false  
irb> fox.is_a? String  
=> true
```

Method nameing convention

# Making Classes and Objects (and throwing them away)

kitt → nil

x → Car:0x63668

Car:0x56a1c Car:061ffc

```
irb --simple-prompt
>> class Car
>> end
=> nil
>> kitt = Car.new
=> #<Car:0x65a1c>
>> x = Car.new
=> #<Car:0x63668>
>> kitt = Car.new
=> #<Car:0x61ffc>
>> kitt
=> #<Car:0x61ffc>
>> kitt = nil
=> nil
>> kitt
=> nil
```

# Instance Variables

Only accessible  
between class and  
end (i.e. within the  
class)

```
$ irb --simple-prompt
>> class Car
>>   @mileage = 0
>> end
=> 0
>> kitt = Car.new
=> #<Car:0x639d8>
>> kitt.@mileage
SyntaxError: compile error
(irb):5: syntax error
      from (irb):5
>>
```

# Access Methods

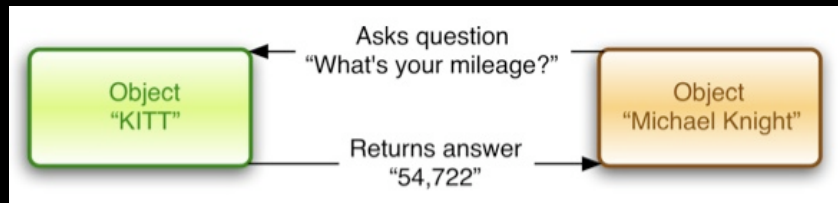
While the instance variables are private to the class, instance methods are what the class presents to the outside world as its interface to manipulate the object.

Method and Message are pretty much interchangeable. The instance method is what responds to the message.

```
$ irb
irb> class Car
irb>   def set_mileage(x)
irb>     @mileage = x
irb>   end
irb>   def get_mileage
irb>     @mileage
irb>   end
irb> end
=> nil
irb> kitt = Car.new
=> #<Car:0x75e54>
irb> kitt.set_mileage(5667)
=> 5667
irb> kitt.get_mileage
=> 5667
```



# Review



```
$ irb
irb> class Car
irb>   def set_mileage(x)
irb>     @mileage = x
irb>   end
irb>   def get_mileage
irb>     @mileage
irb>   end
irb> end
=> nil
irb> kitt = Car.new
=> #<Car:0x75e54>
irb> kitt.set_mileage(5667)
=> 5667
irb> kitt.get_mileage
=> 5667
```

# Accessor Naming Pattern

This makes the syntax to call these methods look more like an assignment statement - makes it very natural to use.

Information hiding.

```
$ irb
irb> class Car
irb>   def mileage=(x)
irb>     @mileage = x
irb>   end
irb>   def mileage
irb>     @mileage
irb>   end
irb> end
=> nil
irb> kitt = Car.new
=> #<Car:0x75e54>
irb> kitt.mileage = 6032
=> 6032
irb> kitt.mileage
=> 6032
```

# Even Simpler

This is a good example of  
a Ruby basic principle:  
“Don’t Repeat Yourself”.

```
$ irb --simple-prompt
>> class Car
>>   attr_accessor :mileage
>> end
=> nil
>> vw = Car.new
=> #<Car:0x63820>
>> vw.mileage = 5
=> 5
>> vw.mileage
=> 5
>>
```

# Class Level Variables

```
$ irb
irb> class Car
irb>   @@number_of_cars = 0
irb>   def initialize
irb>     @@number_of_cars = @@number_of_cars + 1
irb>   end
irb> end
=> nil
```

Initialize method is part of the “new” operation. It is called after the object is created and before the object is returned to the caller.

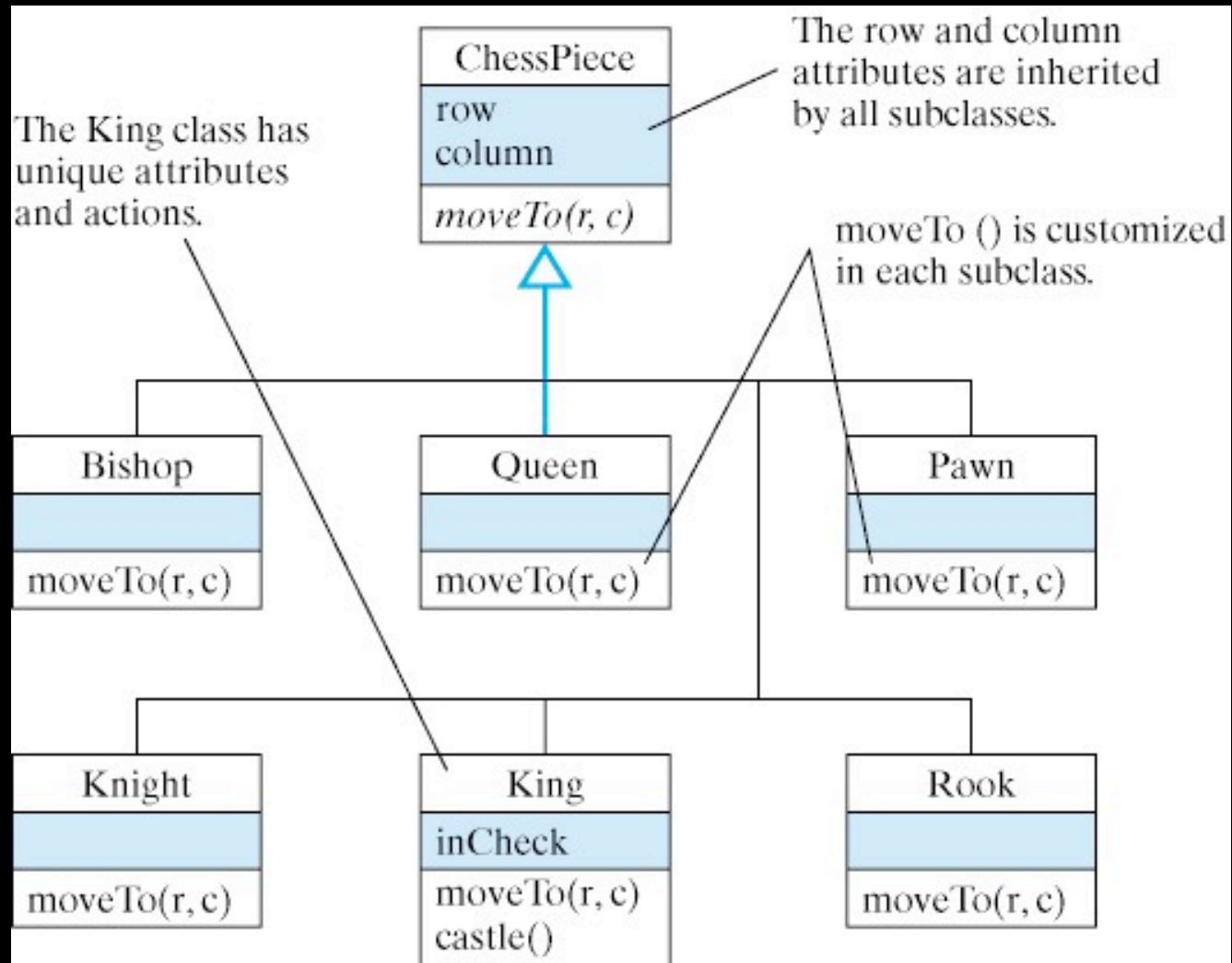
```
$ irb
irb> class Car
irb>   @@number_of_cars = 0
irb>   def self.count
irb>     @@number_of_cars
irb>   end
irb>   def initialize
irb>     @@number_of_cars+=1
irb>   end
irb> end
=> nil
```

```
irb> kitt = Car.new
=> #<0xba8c>
irb> herbie = Car.new
=> #<0x8cd20>
irb> batmobile = Car.new
=> #<0x872e4>
irb> Car.count
=> 3
```

Note - we asked the  
class - not the  
objects.

# Inheritance

- Inheritance is an essential element of the Object Oriented approach
- It is how we create very powerful objects by slightly extending an already powerful object
- It is a form of code reuse
- It is a form of “Don’t Repeat Yourself”
- Typing the same code in multiple places leads to brittle code when you start making changes to the code



# Simple but Powerful

```
class AssnIController < ApplicationController
  def index
    logger.info("Welcome to the index action")
  end
end
```

We write five lines of code and inherit significant amount of functionality that Rails has built into ApplicationController. We don't even know - nor generally care what that functionality we are inheriting. But we can know if we like.

<http://api.rubyonrails.org/classes/ActionController/Base.html>



# Return Values

- The return value is simply the variable or constant that is the last statement in a method

```
$ irb --simple-prompt
>> def method
>>   x = 1
>>   x = x + 1
>>   "Last statement"
>> end
=> nil
>> method
=> "Last statement"
>>
```

# Core Classes / Data Structures

- Arrays - Linear list of items
- Hashes - Set of key / value pairs - fast look up by key
- Strings - quite powerful

# Arrays - Object Style

```
irb> service_mileage = [5000, 15000, 30000, 60000, 100000]
=> [5000, 15000, 30000, 60000, 100000]
irb> service_mileage[0]
=> 5000
irb> service_mileage[2]
=> 30000
irb> available_colors = %w( red green blue black )
=> ["red", "green", "blue", "black"]
irb> available_colors[0]
=> "red"
irb> available_colors[3]
=> "black"
irb> available_colors.empty?
=> false
irb> available_colors.size
=> 4
irb> available_colors.first
=> "red"
irb> available_colors.last
=> "black"
irb> available_colors.delete "red"
=> "red"
irb> available_colors
=> ["green", "blue", "black"]
```

available\_colors

[0]	"red"
[1]	"green"
[2]	"blue"
[3]	"black"

Search through the array and find all the locations containing the value "red" and delete those location, shuffling all of the rows in the array below that item up one.

<http://www.ruby-doc.org/core/classes/Array.html>

# Hashes

```
irb> car_colors = {  
irb>  'kitt' => 'black',  
irb>  'herbie' => 'white',  
irb>  'batmobile' => 'black',  
irb>  'larry' => 'green'  
irb> }  
=> {"kitt"=>"black", "herbie"=>"white",  
    "batmobile"=>"black", "larry"=>"green"}  
irb> car_colors['kitt']  
=> "black"  
irb> car_colors.empty?  
=> false  
irb> car_colors.size  
=> 4  
irb> car_colors.keys  
=> ["kitt", "herbie", "larry", "batmobile"]  
irb> car_colors.values  
=> ["black", "white", "green", "black"]  
irb> car_colors.delete("larry")
```

car\_colors

["kitt"]	"black"
["herbie"]	"white"
["bat.."]	"black"
["larry"]	"green"

Hashes are *\*fast\** - we use hashes to pass in a set of properties as a single parameter.

<http://www.ruby-doc.org/core/classes/Hash.html>

# Strings

```
irb> a_phrase = "The quick brown fox"
=> "The quick brown fox"
irb> a_phrase.class
=> String
irb> 'I\'m a quick brown fox'
=> "I'm a quick brown fox"
irb> "Arnie said, \"I'm back!\""
=> "Arnie said, \"I'm back!\""
irb> %Q(Arnie said, "I'm back!")
=> "Arnie said, \"I'm back!\""
irb> "The current time is: #{Time.now}"
=> "The current time is: Wed Aug 02 21:15:19 CEST
2006"
irb> "The quick brown fox".gsub('fox', 'dog')
=> "The quick brown dog"
rb> "The quick brown fox".include?('fox')
=> true
irb> "The quick brown fox".length
=> 19
irb> "The quick brown fox".slice(0, 3)
=> "The"
```

[http://www.ruby-doc.org/  
core/classes/String.html](http://www.ruby-doc.org/core/classes/String.html)

# Numerics

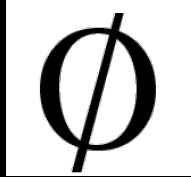
```
irb> 123.class  
=> Fixnum  
irb> 12.5.class=> Float  
irb> 123.integer?  
=> true  
irb> 12.5.integer?  
=> false  
irb> 12.3.round  
=> 12  
irb> 38.8.round  
=> 39  
rb> 0.zero?  
=> true  
irb> 8.zero?  
=> false  
irb> 12.to_f  
=> 12.0  
irb> 11.3.to_i  
=> 11
```

# Symbols

```
irb> :fox  
=> :fox  
irb> :fox.class  
=> Symbol
```

```
irb> car_colors = {  
  :kitt => 'black',  
  :herbie => 'white',  
  :larry => 'green',  
  :batmobile => 'black'  
}  
=> {:kitt=>"black", :herbie=>"white",  
   :larry=>"green", :batmobile=>"black"}  
irb> "fox".to_sym  
=> :fox  
irb> :fox.to_s  
=> "fox"
```

- More efficient than strings when used as hash keys
- Kind of like “constant strings” or “very simple strings”
- Commonly used throughout Ruby



# nil

- It is used to indicate the “lack of an object”
- There is no object here
- When a method wants to return an object but it did not find one - it returns nil
- In asn assignment it also destroys an object
- Zip, Zilch, Nada

```
>> x = String.new("Lets make a string")  
=> "Lets make a string"  
>> x  
=> "Lets make a string"  
>> x = nil  
=> nil  
>> y = String.new("Hello")  
=> "Hello"  
>> z = y  
=> "Hello"  
>> y = nil  
=> nil  
>> z  
=> "Hello"  
>> z = nil  
=> nil
```



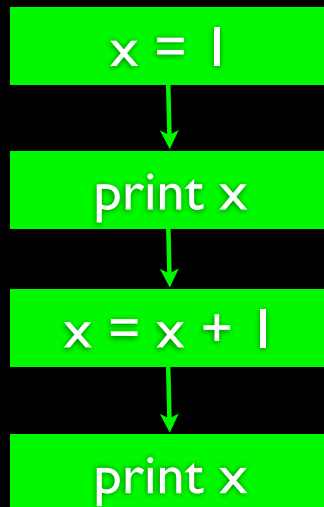
<http://www.ruby-doc.org/core/>

- Time / Data
  - Bignum - For integers  $> 2$  Billion
  - Complex - For imaginary numbers
  - Rational - Ratio of two integers
  - Regexp - Regular expression
  - IPAddr - IP address
  - Set - Like a Hash - but with keys only ( Hash  $<$  Set )
  - Vector - Like an Array - but mathematical
  - Matrix - Two dimensional Array
- there are even more...

# Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a sequence of steps to be done in order
- Some steps are conditional - they may be skipped
- Sometimes a step or group of steps are to be repeated
- Sometimes we store a set of steps to be used over and over as needed several places throughout the program

# Sequential Steps



Program:

```
x = 1
print x
x = x + 1
print x
```

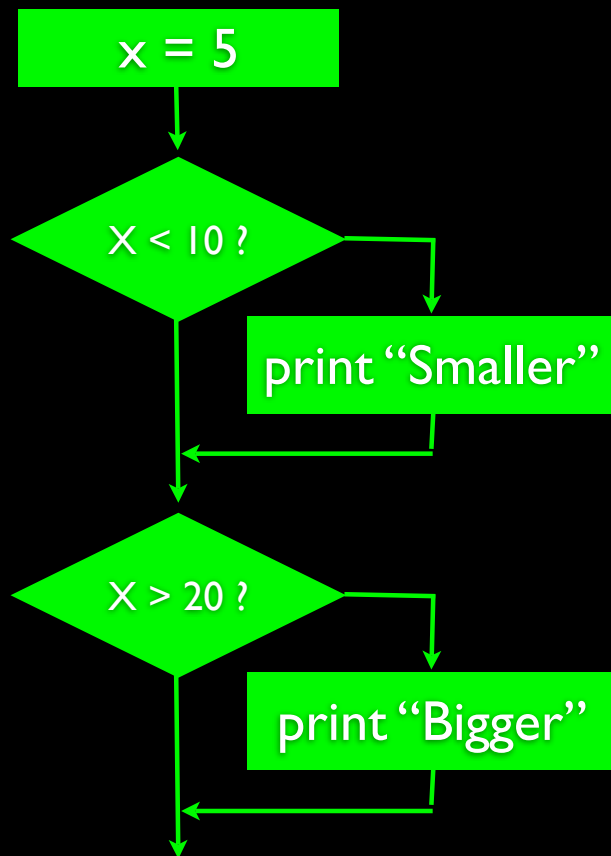
Output:

1

2

When a program is running, it flows from one step to the next. We as programmers set up “paths” for the program to follow.

# Conditional Steps



Program:

$x = 5$

if  $x < 10$

    print "Smaller"

end

if  $x > 20$

    print "Bigger"

end

Output:

Smaller

# If Examples

```
if Car.count == 0  
  puts "No cars have been produced yet."  
end
```

```
if Car.count == 0  
  puts "No cars have been produced yet."  
else  
  puts "New cars can still be produced."  
end
```

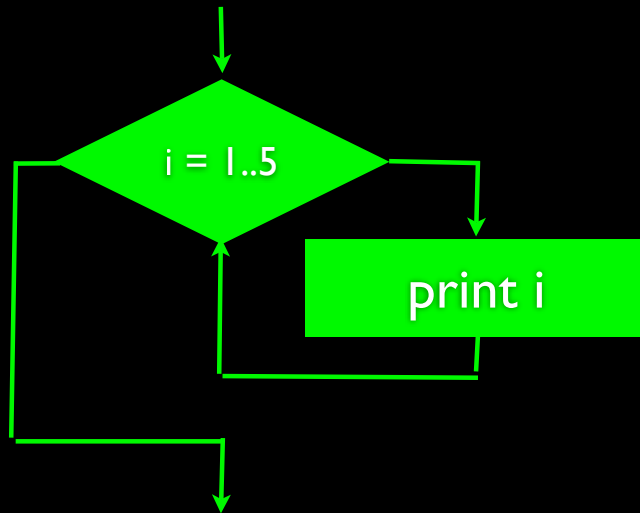
```
if Car.count == 0  
  puts "No cars have been produced yet."  
elsif Car.count >= 10  
  puts "Production capacity has been reached."  
else  
  puts "New cars can still be produced."  
end
```

```
puts "No cars have been produced yet." if Car.count == 0
```

Logic Operators  
< > <= >= != ==

Note that the comparison  
operator for equality is  
"==" not "=".

# Repeated Steps



Program:

```
for i in 1..5  
  print i  
end
```

Output:

```
graph LR; Program[Program] --> Output[Output];
```

1  
2  
3  
4  
5

Two orange arrows point from the `print i` line of the program to the first and last lines of the output, indicating the range of iterations.

# Counted Loops

```
for x in [ kitt, herbie, batmobile, larry ]  
  puts x.mileage  
end
```

```
[ kitt, herbie, batmobile, larry ].each do |x|  
  puts x.mileage  
end
```

```
car_colors = {  
  'kitt' => 'black',  
  'herbie' => 'white',  
  'batmobile' => 'black',  
  'larry' => 'green'  
}
```

```
car_colors.each do |car_name, color|  
  puts "#{car_name} is #{color}"  
end
```

kitt is black  
herbie is white  
batmobile is black  
larry is green

```
10.times { Car.new }  
puts "#{Car.count} cars have been produced."
```

10 cars have been produced.

```
5.upto(7) { |i| puts i }
```

5  
6  
7

Back to Ruby on Rails...



```
class GuessController < ApplicationController

  def index
  end

  def try
    logger.info(params[:guess])
    if params[:guess] == "food"
      @result = "correct"
    else
      @result = "incorrect"
    end
  end
end

end
```

## Inherited Method

```
<meta http-equiv="Content-Type"  
      content="text/html; charset=utf-8" />  
<%= stylesheet_link_tag "style.css" %>
```

```
class AssnIController < ApplicationController  
  
end
```

# Assignment 7A

- Not graded
- Retrieve the list of members in the table from the database
- Due Tuesday March 18 at midnight

## Welcome to SI539

- About
- Contact
- Pictures
- **Membership**
- Application

### SI539 Membership List

These are the members of SI539 who are in good standing and have paid all their dues.

ID	Name	E-Mail
1	Chuck	csev
2	Joe	joe@umich.edu
3	Jane	jane@umich.edu
4	Barcelona	esp@umich.edu
5	Bob	bob@kldjklds

If you think there is an error in the above send mail to [membership@si539.com](mailto:membership@si539.com).

```
class OneController < ApplicationController
  def members
    @lleida = Member.find(:all)
    logger.info "Members method"
    logger.info @lleida
  end

  def thanks
    logger.info "Welcome to the thanks action in the controller"
    logger.info params[:yourname]
    logger.info params[:yourmail]
    memb = Member.create()
    memb.name = params[:yourname]
    memb.email = params[:yourmail]
    memb.save
    @barcelona = memb.id
  end
end
```

## members.rhtml excerpt

```
<table>
  <tr>
    <th>ID</th>
    <th>Name</th>
    <th>E-Mail</th>
  </tr>
  <% @lleida.each do |valencia| %>
    <tr>
      <td><%= valencia.id %></td>
      <td><%= valencia.name %></td>
      <td><%= valencia.email %></td>
    </tr>
  <% end %>
</table>
```

Demo time...

# Summary

- This is not something to memorize
- Have good sources of documentation - and then start coding - learn, experiment ask questions
  - <http://www.ruby-doc.org/>
  - Google: ruby fixnum class
- Work with sample code